REMARKS

Several editorial corrections have been made to the specification. Claims 1 - 18 have been amended. Claims 19 - 24 have been added. No new matter has been introduced. Claims 1 - 24 are now in the application.

I.   Rejection under 35 U.S.C. §102(e)

Page 2 of the Office Action dated December 19, 2002 (hereinafter, "the Office Action") states that Claim 17 has been rejected under 35 U.S.C. §102(e) as being anticipated by Gorelik et al., U. S. Patent Publication US2002/0004799. This rejection is respectfully traversed with reference to Claim 17 as amended herein.

Gorelik's technique creates two copies of the database. See lines 4 - 7 of the Abstract; lines 2 - 3 of paragraph 8; lines 5 - 7 of paragraph 22; lines 3 - 4 of paragraph 30; and lines 6 - 7 of paragraph 39. Applicants' invention does not create duplicate copies of the underlying data, but rather creates two copies of an _index_ to this data. See page 18, lines 6 - 10 of Applicants' specification (using entries in a network routing table as an example of the underlying data, and trees for the index to this data). As will be obvious, using two indexes (as in Applicants' invention) rather than duplicating the underlying data (as in Gorelik's technique) provides significant advantages. Claim 17 now explicitly refers to the indexes.

Applicants' invention creates a serialized record of how each update affects the second index, and after switching the indexes, uses this information to bring the index that was being

used for searches into alignment with the index that was just updated. See p. 17, line 19 - p. 18, line 5. See also p. 21, line18 - p. 22, line 11. This efficient update synchronization technique is specified in the second and fourth elements of Applicants' amended Claim 17.

Applicants' amended Claim 17 specifies, in the third element, that the switch occurs after each update. See p. 17, lines 14 - 16 and p. 22, lines 12 - 13 and lines 17 - 18 of Applicants' specification. Gorelik's technique does not require the switch to occur after every update. Instead, lines 1 - 3 of paragraph 32 state that the switch is delayed until "all loading" is complete. This is different from switching after every update.

Furthermore, Gorelik's switch is triggered differently than the switch in Applicants' claimed invention. The third limitation of Applicants' Claim 17 specifies that the switch is responsive to each update. Gorelik teaches that switches may be triggered by (1) manual intervention of a user, (2) completion of "all of the relevant data flows" of a job, or (3) a switch that is scheduled "at the optimal times". See paragraphs 33, 36, and 37.

As will be obvious, Applicants' automatic per-update switching is much more reliable and timely than putting responsibility on a user to decide when to request a manual switching (see the last two lines of paragraph 37). Furthermore, it is unrealistic to expect that a user could intervene quickly enough to switch after *every* update.

Applicants' per-update switching allows more timely representation of new data than

using Gorelik's "completion of all relevant data flows" approach. See, for example, the sample of buffered data in Gorelik's Appendix A, where records for 13 different employees (having an employee number field, or "EMPNO", set to 7369, 7499, 7521, 7566, etc.) are buffered at one time. Gorelik also states that this "application initiated switching" approach may result in "abandoning" some switches. See lines 7 - 9 of paragraph 37. Applicants' claimed invention does not allow for abandoning switches.

In Gorelik's "schedule switches at the optimal times" approach, paragraph 37 states that "jobs are scheduled to run at specific times and the switch is scheduled for a time when jobs are not scheduled". See lines 9 - 11. This is indicative of a "batch mode" of processing, where a large amount of information might be loaded into a database (i.e., as a "job"), and then this loaded database is switched over to become the "live" database after the job finishes. (See also paragraph 43, which refers to moving update data "file by file"or "several files at a time" to the live database.) As will be obvious, long periods of time may result between synchronizing the two databases when triggering the switches in this manner. This approach is clearly distinct from Applicants' invention, where searches and updates are not required to be "scheduled" or separated in time, and do not proceed in batch mode. Instead, in Applicants' invention, searches may occur simultaneously with an update, and the switch then occurs responsive to that update, allowing the searches (and another update) to continue.

Thus, it can be seen that Gorelik's technique is distinct from Applicants' amended Claim 17. The Examiner is therefore respectfully requested to withdraw the §102(e) rejection.

II.   Rejection under 35 U.S.C. §103(a)

Page 3 of the Office Action states that Claims 1, 3 - 6, 8 - 11, 13 - 16, and 18 have been

rejected under 35 U.S.C. §103(a) as being unpatentable over Gorelik et al., in view of Edwards et

al., U. S. Patent 6,353,820. Page 6 of the Office Action states that Claims 2, 7, and 12 have been

rejected under 35 U.S.C. §103(a) as being unpatentable over Gorelik et al., in view of Edwards et

al., and further in view of Bretl et al., U. S. Patent 6,360,219. These rejections are respectfully

traversed with reference to the claims as amended herein.

Applicants' independent Claims 1, 6, and 11 have been amended to specify that two

indexes are created, rather than two tree structures. (See also section III, Added Claims, below.)

As described above, Gorelik uses two copies of a database, and this is quite different from using

two indexes to the same underlying data. Applicants' independent claims have also been

amended to explicitly specify that the switch is responsive to the update; as described above, this

automatic per-update switching is in contrast to the triggering mechanisms Gorelik describes.

Thus, Gorelik cannot be combined with Edwards to render the independent claims of Applicants'

invention unpatentable.

With reference to Claims 3, 8, and 13, these claims have been amended herein to specify

use of atomic instructions. This amendment is supported in the specification as originally filed.

See, for example, p. 14, lines 12 - 13, and the last sentence of the Abstract. The cited paragraph

from Gorelik has no teaching of atomic instructions.

Rather than using atomic instructions to maintain synchronization between the indexes, Gorelik uses a number of states for his databases (such as Load, Delay, Switch, Delay, and Reconcile). Few details are provided to explain how Gorelik's system "knows" when to transition from one state to another, except that the length of the Delay state can be "user settable" (paragraph 32) or "long enough to ensure that all connections ... are complete" (paragraph 34), and that the Switch state can be triggered in several ways (see paragraphs 36 and 37, which have been contrasted to Applicants' claimed invention in the discussion above). While state transitions may function adequately with user-settable delays in Gorelik's batch-mode update scenario, where the Load state may continue for a considerable time period as a large amount of data is loaded into the load database, this is clearly distinct from Applicants' per-update (i.e., automatic) switching approach.

Applicants' claimed per-update switching ensures that there will always be an index accessible for searching. Gorelik's technique, on the other hand, may result in situations where conflict resolution (or "reconciliation") must be performed before either database can be used. See paragraph 46, which describes an application writing to both databases while a separate update process writes to one of the databases. Paragraphs 47 - 49 provide further details of how these conflicts might be resolved, using timestamps. Problems encountered when trying to synchronize clocks between multiple processes (e.g., the applications and the update process) are well known, and the overhead involved with checking timestamps is readily apparent. The novel techniques of Applicants' invention avoid these problems. (See p. 8, lines 8 - 10 and p. 15, lines 15 - 16, where this is stated.)

As has been demonstrated, Applicants' independent claims are patentably distinct from Gorelik, whether taken singly or in combination with Edwards. Thus, the dependent claims are also deemed patentable. The Examiner is therefore respectfully requested to withdraw the §103(a) rejection.
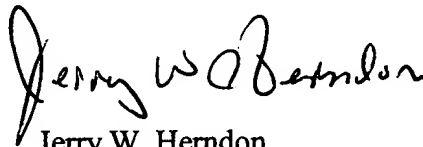
III.   Added Claims

Claims 19 - 24 specify that the indexes of the independent claims may be trees or hash tables. See p. 14, lines 5 - 9, where this is stated in the application as originally filed. Thus, it can be seen that no new matter is introduced with these added claims.

IV.   Conclusion

In conclusion, the stated grounds for rejection have been addressed by amendment and differentiation of the rejected claims from the cited references. Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached pages are captioned "**Version with Markings to Show Changes Made**". Applicants respectfully request reconsideration of the pending rejected claims, withdrawal of all presently outstanding rejections, and allowance of all claims at an early date.

Respectfully submitted,

Jerry W. Herndon
Attorney for Applicants
Reg. No. 27,901

/mld
Docket RSW919990130US1
Phone: 919-543-3754
Fax: 919-254-4330

VERSION WITH MARKINGS TO SHOW CHANGES MADE

In the Specification:

The paragraph on Page 15, lines 2 - 11, has been amended as follows:

-- The LM instruction requires three parameters. The first parameter is a register to be used as the starting location of the load operation, and the second is a register to be used as the ending location. The third parameter points to the beginning storage location of the value(s) to be loaded into the registers. Both the CD and CDS comparison instructions require three parameters. The first and [second] third parameters are pointers to the values being compared. The [third] second parameter is a pointer to a new value that will be copied in place of the value addressed by the [second] first pointer if the first and [second] third values are equal; otherwise, when the values are not equal, the value pointed to by the [second] third pointer is copied into the location addressed by the first pointer. (Note that Compare and Swap is the technique mentioned in the previously-discussed IBM Technical Disclosure Bulletin article for performing atomic updates on AVL trees.) --

In the Claims:

Claim 1 has been amended as follows:

1   1.   A computer program product for serializing data [structure] retrievals and updates, the

2   computer program product embodied on one or more computer-readable media and comprising:

3   computer-readable program code means for creating two identical [tree structures]:

4   indexes, each representing an initial state for accessing stored data;

5   computer-readable program code means for performing searches against a first of the two

6 [trees] <u>indexes</u>;

7 computer-readable program code means for performing a first update against a second of

8 the two [trees] <u>indexes</u>, yielding a revised [tree] <u>index</u>;

9 <u>computer-readable program code means for serializing information on how the first</u>

10 <u>update affected the second index</u>;

11 computer-readable program code means for switching the first [tree] <u>index</u> and the

12 revised [tree] <u>index, responsive to operation of the computer-readable program code means for</u>

13 <u>performing the first update</u>, such that the first [tree] <u>index</u> becomes the second [tree] <u>index</u> and

14 the revised [tree] <u>index</u> becomes the first [tree] <u>index</u>;

15 computer-readable program code means for [performing] <u>applying</u>, after operation of the

16 computer-readable program code means for switching, [a second update against] <u>the serialized</u>

17 <u>information to</u> the second [tree] <u>index</u>, yielding a [synchronized tree] <u>second index</u> that is

18 <u>synchronized with, and</u> structurally identical to, the first [tree] <u>index</u>; and

19 computer-readable program code means for performing subsequent searches against the

20 first [tree] <u>index</u>.

Claim 2 has been amended as follows:

1 2. The computer program product according to Claim 1, further comprising:

2 computer-readable program code means for obtaining an exclusive lock <u>on the second</u>

3 <u>index</u> prior to operation of the computer-readable program code means for performing the first

4 update; and

5 computer-readable program code means for releasing the exclusive lock after operation of

6   the computer-readable program code means for [performing the second update] applying the

7   serialized information [and the computer-readable program code means for switching].

Claim 3 has been amended as follows:

1   3.   The computer program product according to Claim 1, wherein atomic [transactions]

2   instructions are used to maintain proper synchronization between the first [tree] index and the

3   second [tree] index.

Claim 4 has been amended as follows:

1   4.   The computer program product according to Claim 1, wherein the computer-readable

2   · program code means for [performing the first update] serializing information further comprises

3   computer-readable program code means for queuing a transaction, and wherein the computer-

4   readable program code means for [performing the second update] applying the serialized

5   information further comprises computer-readable program code means for applying the queued

6   transaction [against] to the second [tree] index [that results from operation of the computer-

7   readable program code means for switching].

Claim 5 has been amended as follows:

1   5.   The computer program product according to Claim 1, further comprising computer-.

2   readable program code means for performing a subsequent update against the [synchronized tree]

3   second index that results from operation of the computer-readable program code means for

4   [performing the second update] applying the serialized information; and wherein operation of the

5   computer-readable program code means for performing the subsequent update causes another

6   operation of the <u>computer-readable program code means for serializing, the</u> computer-readable

7   program code means for switching, <u>and the computer-readable program code means for applying</u>.

Claim 6 has been amended as follows:

1   6.   A system for serializing data [structure] retrievals and updates in a computing

2   environment, comprising:

3   means for creating two identical [tree structures] <u>indexes</u>, each representing an initial

4   state for accessing stored data;

5   means for performing searches against a first of the two [trees] <u>indexes</u>;

6   means for performing a first update against a second of the two [trees] <u>indexes</u>, yielding a

7   revised [tree] <u>index</u>;

8   <u>means for serializing information on how the first update affected the second index;</u>

9   means for switching the first [tree] <u>index</u> and the revised [tree] <u>index, responsive to</u>

10   <u>operation of the means for performing the first update</u>, such that the first [tree] <u>index</u> becomes

11   the second [tree] <u>index</u> and the revised [tree] <u>index</u> becomes the first [tree] <u>index</u>;

12   means for [performing] <u>applying</u>, after operation of the means for switching, [a second

13   update against] <u>the serialized information to</u> the second [tree] <u>index</u>, yielding a [synchronized

14   tree] <u>second index</u> that is <u>synchronized with, and</u> structurally identical to, the first [tree] <u>index</u>;

15   and

16   means for performing subsequent searches against the first [tree] <u>index</u>.

Claim 7 has been amended as follows:

1   7.   The system according to Claim 6, further comprising:

2   means for obtaining an exclusive lock on the second index prior to operation of the

3   means for performing the first update; and

4   means for releasing the exclusive lock after operation of the means for [performing the

5   second update] applying the serialized information [and the means for switching].

Claim 8 has been amended as follows:

1   8.   The system according to Claim 6, wherein atomic [transactions] instructions are used to

2   maintain proper synchronization between the first [tree] index and the second [tree] index.

Claim 9 has been amended as follows:

1   9.   The system according to Claim 6, wherein the means for [performing the first update]

2   serializing information further comprises means for queuing a transaction, and wherein the

3   means for [performing the second update] applying the serialized information further comprises

4   means for applying the queued transaction [against] to the second [tree] index [that results from

5   operation of the means for switching].

Claim 10 has been amended as follows:

1   10.   The system according to Claim 6, further comprising means for performing a subsequent

2   update against the [synchronized tree] second index that results from operation of the means for

3   [performing the second update] applying the serialized information; and wherein operation of the

4 means for performing the subsequent update causes another operation of the means for

5 serializing, the means for switching, and the means for applying.

Claim 11 has been amended as follows:

1 11.   A method for serializing data [structure] retrievals and updates in a computing

2 environment, comprising steps of:

3   creating two identical [tree structures] indexes, each representing an initial state for

4 accessing stored data;

5   performing searches against a first of the two [trees] indexes;

6   performing a first update against a second of the two [trees] indexes, yielding a revised

7 · [tree] index;

8   serializing information on how the first update affected the second index;

9   switching the first [tree] index and the revised [tree] index, responsive to performing the

10 first update, such that the first [tree] index becomes the second [tree] index and the revised [tree]

11 index becomes the first [tree] index;

12   [performing] applying, after the switching step, [a second update against] the serialized

13 information to the second [tree] index, yielding a second index [synchronized tree] that is

14 synchronized with, and structurally identical to, the first [tree] index; and

15   performing subsequent searches against the first [tree] index.

Claim 12 has been amended as follows:

1 12.   The method according to Claim 11, further comprising steps of:

2   obtaining an exclusive lock <u>on the second index</u> prior to performing the first update; and

3   releasing the exclusive lock after <u>applying the serialized information</u> [performing the

4 second update and the switching].

Claim 13 has been amended as follows:

1   13.   The method according to Claim 11, wherein atomic [transactions] <u>instructions</u> are used to

2 maintain proper synchronization between the first [tree] <u>index</u> and the second [tree] <u>index</u>.

Claim 14 has been amended as follows:

1   14.   The method according to Claim 11, wherein the step of [performing the first update]

2 <u>serializing information</u> further comprises <u>the step of</u> queuing a transaction, and wherein the step

3 of [performing the second update] <u>applying the serialized information</u> further comprises <u>the step</u>

4 <u>of</u> applying the queued transaction [against] <u>to</u> the second [tree] <u>index</u> [that results from

5 operation of the switching step].

Claim 15 has been amended as follows:

1   15.   The method according to Claim 11, further comprising the step of performing a

2 subsequent update against the [synchronized tree] <u>second index</u> that results from [performing the

3 second update] <u>applying the serialized information</u>; and wherein the step of performing the

4 subsequent update causes repeating the <u>serializing,</u> switching<u>, and applying</u> step<u>s</u>.

Claim 16 has been amended as follows:

1   16.   A method of serializing access to data [structures] in a computing system, comprising

2   steps of:

3   maintaining two trees as indexes to the data, a first of which is used for [one or more

4   concurrent] searches and a second of which is used for [an] update operations;

5   serializing, for each update operation, a record of how the update operation affected the

6   second tree;

7   switching the two trees, responsive to [after] performing [the] each update operation; and

8   [synchronizing] applying the serialized record to the newly-switched second tree, [the

9   two trees] such that both the first tree and the second tree reflect the update operation.

.   Claim 17 has been amended as follows:

1   17.   A method of serializing access to data [structures] in a computing system, comprising

2   steps of:

3   maintaining two indexes to the data [data structures], a first of which is used for [one or

4   more concurrent] searches and a second of which is used for [an] update operations;

5   serializing, for each update operation, a record of how the update operation affected the

6   second index;

7   switching the two indexes, responsive to [data structures after] performing [the] each

8   update operation; and

9   [synchronizing the two data structures] applying the serialized record to the newly-

10 switched second index, such that both the first index and the second index reflect the update

11 operation.

Claim 18 has been amended as follows:

1   18.   The method [of] <u>according to</u> Claim 17, wherein the two [data structures] <u>indexes</u> are B-

2   trees.

The following new Claims 19 - 24 have been added:

1   19.   The computer program product according to Claim 1, wherein the indexes are

2   implemented as trees.

1   20.   The computer program product according to Claim 1, wherein the indexes are

2   implemented as hash tables.

1   21.   The system according to Claim 6, wherein the indexes are implemented as trees.

1   22.   The system according to Claim 6, wherein the indexes are implemented as hash tables.

1   23.   The method according to Claim 11, wherein the indexes are implemented as trees.

1   24.   The method according to Claim 11, wherein the indexes are implemented as hash tables.